

An Algorithmic Approach to the Asynchronous Computability Theorem

Vikram Saraph

Maurice Herlihy

Eli Gafni

March 27, 2017

Abstract

The asynchronous computability theorem (ACT) uses concepts from combinatorial topology to characterize which tasks have wait-free solutions in read-write memory. A task can be expressed as a relation between two chromatic simplicial complexes. The theorem states that a task has a protocol (algorithm) if and only if there is a certain chromatic simplicial map compatible with that relation.

While the original proof of the ACT relied on an involved combinatorial argument, Borowsky and Gafni later proposed an alternative proof that relied on an algorithmic construction, termed the “convergence algorithm”. The description of this algorithm was incomplete, and presented without proof. In this paper, we give the first complete description, along with a proof of correctness.

1 Introduction

Herlihy and Shavit’s [13, 12] *asynchronous computability theorem* (ACT) characterizes which tasks are solvable in asynchronous shared-memory systems. Informally, this characterization employs the language of combinatorial topology: a task can be expressed as a relation between two “colored” simplicial complexes. The theorem states that a protocol (algorithm) exists if and only if there is a “color-preserving” simplicial map from one complex to the other compatible with the task’s relation. This approach replaces the usual operational model, where computations unfold in time, with a static model in which all possible protocol interleavings are captured in a single combinatorial structure.

The proof of the ACT contains one difficult step. Using the classical *simplicial approximation theorem* [15, p.89], it is straightforward to construct a simplicial map having all desired properties except that of being color-preserving. To make this map color-preserving required a rather long construction employing mechanisms from point-set topology, such as ϵ -balls and Cauchy sequences.

Borowsky and Gafni [5] later proposed an alternative proof strategy for the ACT in which the essential chromatic property was guaranteed by an algorithm, rather than by a combinatorial construction. The description of this algorithm was sketchy, however, and no proof was provided. In this paper, we give the first complete description of their algorithm, along with its first proof of correctness.

The paper is structured as follows. In Section 2, we briefly introduce the combinatorial model and the relevant topological machinery, and we give a short exposition on the work by Borowsky and Gafni [5]. In Section 3, we describe the convergence algorithm and a subroutine used by it. In Section 4, we give a complete proof of correctness of the convergence algorithm. In Section 5, we explain how the convergence algorithm can be applied to more general tasks. In Section 6, we summarize some related work, and in Section 7, we conclude with some final remarks.

2 Combinatorial Topology

This section reviews the basic notions of combinatorial topology that will be used to describe shared-memory computation. Our basic construct, called a simplicial complex, can be defined in two complementary ways: combinatorial and geometric. Sometimes it is convenient to use one view, sometimes the other, and we will go back and forth as needed.

2.1 Abstract Simplicial Complexes

Given a finite set V , and a family \mathcal{K} of finite subsets of V , we say that \mathcal{K} is an *abstract simplicial complex* on V if the following hold:

- (1) if $X \in \mathcal{K}$, and $Y \subseteq X$, then $Y \in \mathcal{K}$;
- (2) $\{v\} \in \mathcal{K}$, for all $v \in V$.

Each set in \mathcal{K} is called a *simplex*, usually denoted by lower-case Greek letters σ and τ . A subset of a simplex is a *face* of that simplex. The *dimension* $\dim \sigma$ is one less than the number of elements of σ , or $|\sigma| - 1$. We use “ n -simplex” as shorthand for “ n -dimensional simplex”, and similarly for “ n -face”. A simplex ϕ in \mathcal{K} is a *facet* of \mathcal{K} if ϕ is not contained in any other simplex. A complex is *pure* if all its facets have the same dimension. The *n -skeleton* of a complex \mathcal{K} , $\text{skel}^n \mathcal{K}$, is the complex formed by all simplexes of \mathcal{K} of dimension n or less. If \mathcal{K} and \mathcal{L} are complexes with $\mathcal{K} \subseteq \mathcal{L}$, we say \mathcal{K} is a *subcomplex* of \mathcal{L} .

We use Δ^n to denote the complex consisting of a single n -simplex and its faces. A *labeling* is a map $\lambda : V \rightarrow D$, where D is an arbitrary domain, such as the natural numbers. A map ϕ carrying vertexes of \mathcal{K} to vertexes of \mathcal{L} is a *simplicial map* if it also carries simplexes of \mathcal{K} to simplexes of \mathcal{L} . A simplicial map $\chi : \mathcal{K} \rightarrow \mathcal{L}$ is a *coloring* if for each simplex σ of \mathcal{K} , the vertexes of σ are mapped to distinct vertexes. Most complexes considered here are endowed with a coloring χ , and such complexes are called *chromatic*. A simplicial map $\phi : \mathcal{K} \rightarrow \mathcal{L}$ is *color-preserving* if $\chi(v) = \chi(\phi(v))$ for vertexes $v \in \mathcal{K}$.

For complexes \mathcal{K} and \mathcal{L} , a *carrier map*, written

$$\Gamma : \mathcal{K} \rightarrow 2^{\mathcal{L}},$$

maps each simplex $\sigma \in \mathcal{K}$ to a subcomplex $\Gamma(\sigma)$ of \mathcal{L} , such that if $\tau \subseteq \sigma$, then $\Gamma(\tau) \subseteq \Gamma(\sigma)$. A simplicial map γ is *carried by* a carrier map Γ if $\gamma(\sigma) \subseteq \Gamma(\sigma)$ for every simplex in their domain. In this case, γ is said to be *carrier-preserving*.

The *star* of a simplex $\sigma \in \mathcal{C}$, written $\text{St}(\sigma, \mathcal{C})$, or $\text{St}(\sigma)$ when \mathcal{C} is clear from context, is the complex that consist of every simplex τ which contains σ , and every simplex contained in such τ .

2.2 Geometric Complexes

In the alternative geometric view, we embed a complex in a Euclidean space \mathbb{R}^d of sufficiently high dimension.

Given a set $X = \{x_0, \dots, x_n\}$ of points in \mathbb{R}^d , their *convex hull*, written $\text{conv } X$, is the set of points y that can be expressed as

$$y = \sum_{i=0}^n t_i \cdot x_i,$$

where the coefficients t_i satisfy $0 \leq t_i \leq 1$, and $\sum_{i=0}^n t_i = 1$. The t_i are called the *barycentric coordinates* of y with respect to X . The set X is *affinely independent* if no point in the set can be expressed as a weighted sum of the others.

A (geometric) vertex is a point in \mathbb{R}^d , and a (geometric) n -simplex is the convex hull of $(n+1)$ affinely-independent geometric vertexes. A *geometric simplicial complex* \mathcal{K} in \mathbb{R}^d is a collection of geometric simplexes, such that

- (1) any face of a $\sigma \in \mathcal{K}$ is also in \mathcal{K} ;
- (2) for all $\sigma, \tau \in \mathcal{K}$, their intersection $\sigma \cap \tau$ is a face of each of them.

We use $|\mathcal{K}|$ to denote the point-set occupied by the geometric complex \mathcal{K} .

Given a geometric simplicial complex \mathcal{K} , we can define the underlying abstract simplicial complex $\mathcal{C}(\mathcal{K})$ as follows: take the union of all the sets of vertexes of the simplexes of \mathcal{K} as the vertexes of $\mathcal{C}(\mathcal{K})$, then for each simplex $\sigma = \text{conv}\{v_0, \dots, v_n\}$ of \mathcal{K} take the set $\{v_0, \dots, v_n\}$ to be a simplex of $\mathcal{C}(\mathcal{K})$.

In the opposite direction, given an abstract simplicial complex \mathcal{A} with finitely many vertexes, there exist many geometric simplicial complexes \mathcal{K} , such that $\mathcal{C}(\mathcal{K}) = \mathcal{A}$.

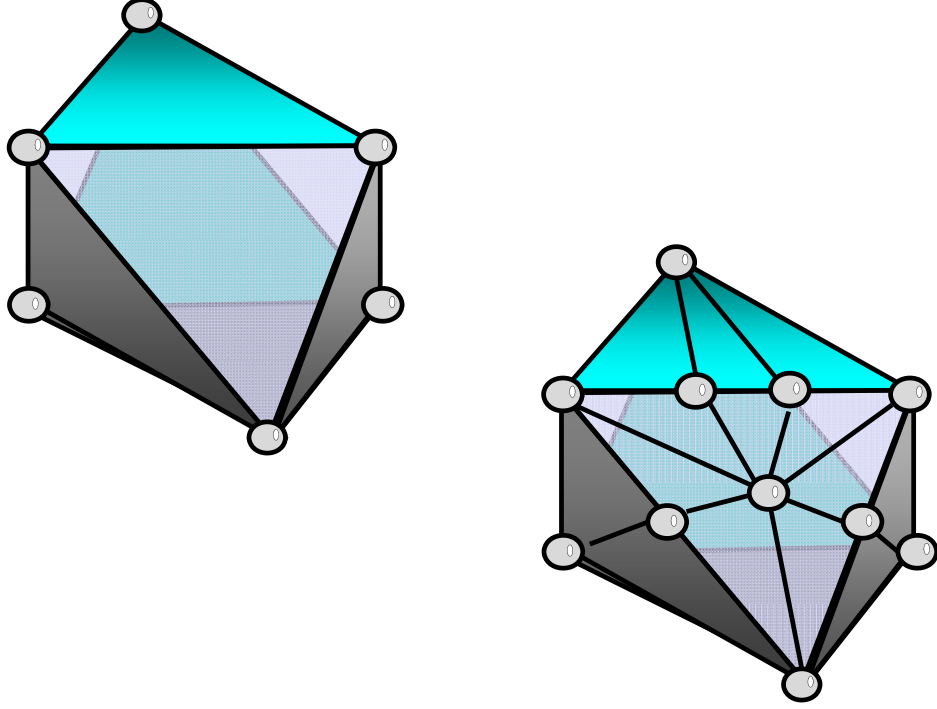


Figure 1: A geometric complex and a subdivision of it.

The *open star*, denoted $\text{St}^\circ(\sigma)$, is the union of the interiors of the simplexes that contain σ :

$$\text{St}^\circ(\sigma) = \bigcup_{\tau \supseteq \sigma} \text{Int } \tau.$$

Note that $\text{St}^\circ(\sigma)$ is not an abstract or geometric simplicial complex, but just a open point-set in \mathcal{C} .

2.3 Subdivisions

If \mathcal{A} and \mathcal{B} are geometric complexes, a continuous map $f : |\mathcal{A}| \rightarrow |\mathcal{B}|$ is *carried by* a carrier map $\Phi : \mathcal{A} \rightarrow 2^{\mathcal{B}}$ if, for every simplex $\sigma \in \mathcal{A}$, $f(\sigma) \subseteq |\Phi(\sigma)|$.

Informally, a *subdivision* of a complex \mathcal{K} is constructed by “dividing” the simplexes of \mathcal{K} into smaller simplexes, to obtain another complex \mathcal{L} . Subdivisions can be defined for both geometric and abstract complexes.

Definition 2.1. A geometric complex \mathcal{B} is called a *subdivision* of a geometric complex \mathcal{A} if the following two conditions are satisfied:

- (1) $|\mathcal{A}| = |\mathcal{B}|$;
- (2) each simplex of \mathcal{A} is the union of finitely many simplexes of \mathcal{B} .

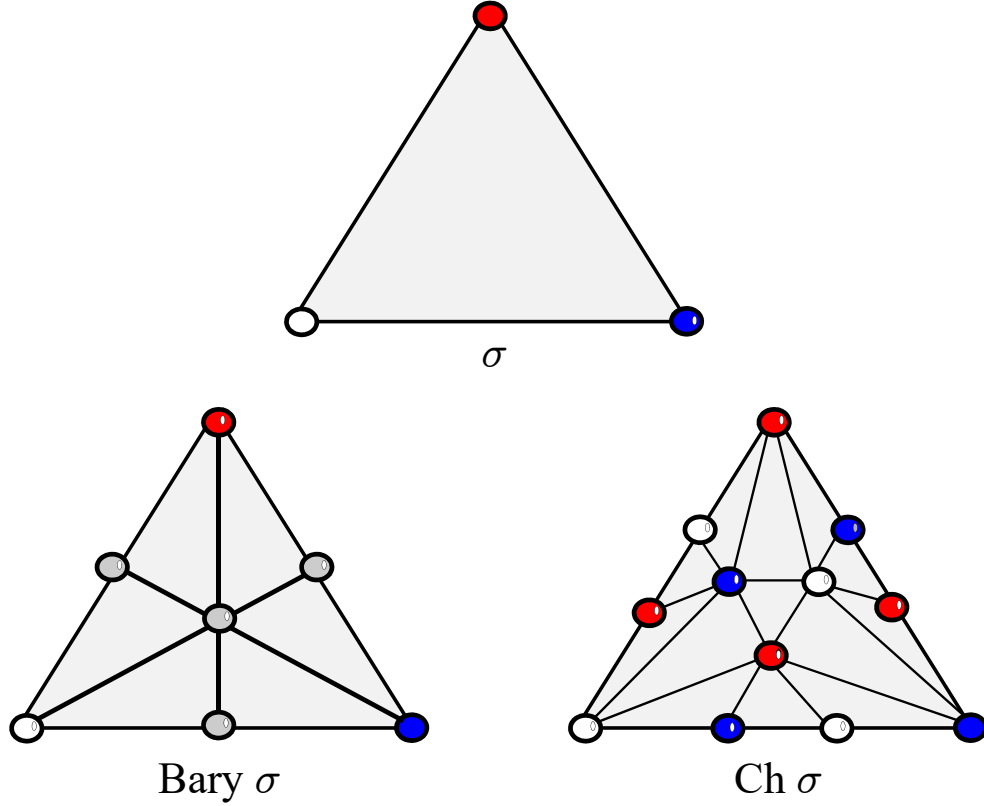


Figure 2: A simplex σ (top), the barycentric subdivision $\text{Bary } \sigma$ (bottom left), and the standard chromatic subdivision $\text{Ch } \sigma$ (bottom right).

Figure 1 shows a geometric complex and a subdivision of that complex.

Subdivisions can be defined for abstract simplicial complexes as well.

Definition 2.2. Let \mathcal{A} and \mathcal{B} be abstract simplicial complexes. We say that \mathcal{B} *subdivides* the complex \mathcal{A} if there exists a homeomorphism $h : |\mathcal{A}| \rightarrow |\mathcal{B}|$ and a carrier map $\Phi : \mathcal{A} \rightarrow 2^{\mathcal{B}}$, such that for every simplex $\sigma \in \mathcal{A}$, the restriction $h|_{|\sigma|}$ is a homeomorphism between $|\sigma|$ and $|\Phi(\sigma)|$.

A subdivision $\text{Div}(\mathcal{K})$ is *chromatic*, if $\text{Div}(\mathcal{K})$ is chromatic, and for every simplex $\sigma \in \mathcal{K}$, $\text{Div}(\sigma)$ and σ have the same colors.

Let $\text{Div } \mathcal{K}$ be a subdivision. The *carrier* of a simplex σ , $\text{Car}(\sigma)$, is the smallest simplex $\kappa \in \text{Div } \mathcal{K}$ such that $\sigma \in \text{Div}(\kappa)$.

2.3.1 Barycentric Subdivision

In classical combinatorial topology, the following barycentric subdivision is perhaps the most widely used.

Definition 2.3. Let \mathcal{K} be an abstract simplicial complex. Its *barycentric* subdivision $\text{Bary } \mathcal{K}$ is the abstract simplicial complex whose vertexes are the non-empty simplexes of \mathcal{K} . A $(k+1)$ -tuple $(\sigma_0, \dots, \sigma_k)$ is a simplex of $\text{Bary } \mathcal{K}$ if and only if the tuple can be indexed so that $\sigma_0 \subset \dots \subset \sigma_k$.

2.3.2 Standard Chromatic Subdivision

The *standard chromatic subdivision* $\text{Ch}(\mathcal{K})$ is the chromatic analog to the barycentric subdivision. (See Figure 2.)

Definition 2.4. Let (\mathcal{A}, χ) be a chromatic abstract simplicial complex. Its *standard chromatic subdivision* $\text{Ch}(\mathcal{A})$ is the abstract simplicial complex whose vertexes have the form (i, σ_i) , where $i \in \{0, \dots, n\}$, σ_i is a non-empty face of σ , and $i \in \chi(\sigma_i)$. A $(k+1)$ -tuple $(\sigma_0, \dots, \sigma_k)$ is a simplex of $\text{Ch}(\mathcal{A})$ if and only if

- the tuple can be indexed so that $\sigma_0 \subseteq \dots \subseteq \sigma_k$.
- for $0 \leq i, j \leq k$, if $i \in \chi(\sigma_j)$ then $\sigma_i \subseteq \sigma_j$.

Finally, to make the subdivision chromatic, we define the coloring $\chi : \text{Ch}(\mathcal{K})$ to be $\chi(i, \sigma) = i$.

This subdivision extends to complexes in the obvious way, and it can be *iterated* to produce subdivisions $\text{Ch}^N(\sigma)$, for $N > 0$.

2.4 Links and Connectivity

Subdivided simplexes exhibit an important property called *link-connectivity*. We provide the necessary definitions below.

Definition 2.5. The *link* of a σ in complex \mathcal{K} , denoted $\text{Lk}(\sigma, \mathcal{K})$, is the subcomplex of \mathcal{K} consisting of all simplexes τ disjoint from σ such that $\tau \cup \sigma \in \mathcal{K}$.

One can think of the link of σ as a simplicial neighborhood that encompasses σ . Next, we present the concept of *connectivity* as taken from algebraic topology.

Definition 2.6. Let S^k denote the k -dimensional sphere. A complex \mathcal{K} is *k-connected* if, for all $0 \leq m \leq k$, any continuous map $f : S^m \rightarrow |\mathcal{K}|$ can be extended to a continuous $F : D^{m+1} \rightarrow |\mathcal{K}|$, where the sphere S^m is the boundary of the disk D^{m+1} .

One way to think about this property is that any map f of the k -sphere that cannot be “filled in” represents a k -dimensional “hole” in the complex.

Definition 2.7. A pure n -dimensional complex \mathcal{K} is *link-connected* if for all $\sigma \in \mathcal{K}$, $\text{Lk}(\sigma, \mathcal{K})$ is $(n - \dim(\sigma) - 2)$ -connected.

Informally, link-connectivity ensures that a complex cannot be “pinched” too thinly. All subdivided simplexes are link-connected, which is an important property for proving the main theorem.

2.5 Useful Theorems

We will need the following *pasting lemma* from point-set topology.

Lemma 2.8. Let X_1 and X_2 be closed sets of a common space, with continuous $f_i : X_i \rightarrow Y$ that coincide on the intersection $X_1 \cap X_2$. Then the function $f : X_1 \cup X_2 \rightarrow Y$, defined in terms of the f_i , is a continuous function.

We will also need the following version of the classical *simplicial approximation theorem*.

Definition 2.9. Let \mathcal{A} and \mathcal{B} be abstract simplicial complexes, let $f : |\mathcal{A}| \rightarrow |\mathcal{B}|$ be a continuous map, and let $\varphi : \mathcal{A} \rightarrow \mathcal{B}$ be a simplicial map. The map φ is called a *simplicial approximation* to f , if for every simplex α in \mathcal{A} we have

$$f(\text{Int } |\alpha|) \subseteq \bigcap_{a \in \alpha} \text{St}^\circ(\varphi(a)) = \text{St}^\circ(\varphi(\alpha)), \quad (2.1)$$

where $\text{Int } |\alpha|$ denotes the interior of $|\alpha|$.

Theorem 2.10. Let \mathcal{A} and \mathcal{B} be simplicial complexes. Given a continuous map $f : |\mathcal{A}| \rightarrow |\mathcal{B}|$, there is an $N > 0$ such that f has a simplicial approximation $\varphi : \text{Ch}^N(\mathcal{A}) \rightarrow \mathcal{B}$.

This theorem remains true if we replace the standard chromatic subdivision $\text{Ch}(\cdot)$ with the barycentric subdivision $\text{Bary}(\cdot)$.

3 Read-Write Memory

In this section we describe and motivate our model of computation. A *distributed system* is a collection of sequential computing entities, called *processes*, that cooperate to solve a problem, called a *task*. The processes communicate by reading and writing a shared memory. Each process runs a program that defines how and when it communicates with other processes. Collectively these programs define a concurrent algorithm, or *protocol*.

The theory of distributed computing is largely about what is computable in the presence of timing uncertainty and failures. Here, we assume processes are subject to *crash failures*, in which a faulty process simply halts and falls silent. We focus on *wait-free* protocols that solve particular tasks when any proper subset of the of processes may fail. We adopt an *asynchronous* timing model, where processes run at arbitrary, unpredictable speeds, and there is no bound on process step time. Note that a failed process cannot be distinguished from a slow process.

3.1 Processes

As stated, a process is an automaton. Each process has a unique *name*. In a computation, it starts with an input value, takes a finite number of steps, and halts with an output value. The task specification defines which values can be assigned as inputs, and which can be accepted as outputs.

A process state is modeled as a vertex, labeled with both that process's name and its state. A system state is a simplex, where each vertex represents a process and its state, and the vertexes represent mutually compatible states. Such is simplex colored by process names, and labeled with process states. An initial system state is represented as a simplex whose set of vertexes represent the simultaneous states of distinct processes. The set of all possible states forms a chromatic complex, colored by process names, and labeled with process states. Henceforth, we speak of a vertex's *color* as shorthand for the name of the corresponding process.

3.2 Tasks

To reduce computation to its simplest form, we consider a fundamental unit of computation called a *task*. An input to a task is distributed: only part of the input is given to each process. The output from a task is also distributed: only part of the output is computed by each process. The task specification states which outputs can be produced in response to each inputs.

A *task* is a triple $(\mathcal{I}, \mathcal{O}, \Gamma)$, where \mathcal{I} is the *input complex* that defines all possible initial configurations, \mathcal{O} is the *output complex* that defines all possible final configurations, and Γ is a carrier map carrying each m -simplex of \mathcal{I} to a subcomplex of m -simplexes of \mathcal{O} , for $0 \leq m \leq n$. Γ has the following interpretation: for each $\sigma \in \mathcal{I}$, if the $(\dim \sigma + 1)$ processes in σ start with the designated input values, and the remaining $n - \dim \sigma$ processes fail without taking any steps, then each simplex in $\Gamma(\sigma)$ corresponds to a legal final state of the non-faulty processes.

A task has the following interpretation. For each simplex $\sigma \in \mathcal{I}$, if the $(\dim \sigma + 1)$ processes in σ each starts on the unique vertex colored with its name, taking as input that vertex's labeled value, and the remaining $n - \dim \sigma$ processes fail without taking any steps, then each simplex in $\Gamma(\sigma)$ corresponds to a legal final state of the non-faulty processes, where each process halts on the unique vertex colored with its name, taking as output that vertex's labeled value.

The most common task considered in this paper is a *convergence* task. We are given a input complex \mathcal{I} colored by process names, and a chromatic subdivision of \mathcal{I} . In the task $(\mathcal{I}, \text{Div}(\mathcal{I}), \text{Div})$, the $n + 1$ processes start on vertexes of an n -simplex σ of \mathcal{I} , where each process's input vertex is colored by that process's name. The processes halt on a single simplex of $\text{Div}(\sigma)$, and each process halts on an vertex is colored by that process's name.

In some circumstances, we relax the requirement that each processes halt on an output vertex labeled with its own name. A *colorless* task is also defined by a triple $(\mathcal{I}, \mathcal{O}, \Gamma)$, but there is no requirement that the decision map be color-preserving.

3.3 Protocols

A *protocol* is a concurrent algorithm to solve a task: initially each process knows its own part of the input, but not the others'. Each process communicates with the others by reading and writing a shared memory, and eventually halts with its own output value. Collectively, the individual output values form the task's output.

Here, we are concerned with protocols where processes communicate by reading and writing shared memory. Without loss of generality, we consider *full-information* protocols in which each process repeatedly writes its current state to the shared memory, and constructs its new state by taking a snapshot of (instantaneously reading) the states written by other processes. After a finite number of such communication rounds, each process applies a *decision map* to its state to choose an output value.

All possible executions of a protocol can be modeled as a chromatic simplicial complex. We think of the protocol executions as a chromatic carrier map Ξ from the input complex \mathcal{I} to \mathcal{P} , the chromatic *protocol complex*. The carrier map Ξ carries each input simplex $\sigma \in \mathcal{I}$ to a subcomplex $\Xi(\sigma)$ of the protocol complex. Ξ carries each $\sigma \in \mathcal{I}$ to the executions in which the processes in σ do not hear from any other process.

The protocol complex is related to the output complex by the decision map δ , a chromatic simplicial map that sends each vertex v in the protocol complex to a vertex w in the output complex, colored with the same process name. The decision value labeling w is the value which the corresponding process takes as its output value. A protocol with carrier map Ξ solves the task if the carrier map obtained by composing δ with Ξ is carried by Δ .

3.4 Read-Write Memory

In the most natural shared-memory model, the processes share an ordered sequence of *registers*, memory units that can hold values from an arbitrary domain. A process can *read* a register, returning its current contents, or it can *write* a new value to that register, obliterating that register's prior value.

There are many variations of the asynchronous read-write model, all computationally equivalent in the sense that one model can simulate the others with overhead polynomial in the number of processes. Here, we will use two distinct alternatives. In the *snapshot* model, each process can write atomically to a single register, but it can atomically read any set of registers, an operation called a *snapshot*.

It is also sometimes convenient to use a variation known as the *immediate snapshot* model [4, 16] An *immediate snapshot* takes place in two contiguous steps. In the first step, a process writes its view to a word in memory, possibly concurrently with other processes. In the very next step, it takes a snapshot of some or all of the memory, possibly concurrently with other processes. It is important to understand that that in an immediate snapshot, the snapshot step takes place *immediately after* the write step.

Although, modern multicores do not support either the snapshot or the immediate snapshot model directly, either model can be simulated by more conventional models, and vice-versa [11, Ch. 14]. The snapshot model is convenient for expressing specific algorithms, while the immediate snapshot model is convenient because the protocol complex generated by an N -round immediate snapshot execution is the subdivision $\text{Ch}^N(\mathcal{I})$ of the input complex [11, Ch. 4]. (Using individual reads and writes, the protocol complexes are not subdivisions, although they can be retracted to subdivisions.)

4 The Asynchronous Computability Theorem

The ACT states that a task $T = (\mathcal{I}, \mathcal{O}, \Gamma)$ has a wait-free protocol in read-write memory exactly when there is a chromatic, simplicial map from a chromatic subdivision of \mathcal{I} to \mathcal{O} carried by Γ .

Theorem 4.1. A task $(\mathcal{I}, \mathcal{O}, \Gamma)$ has a wait-free read-write protocol if and only if there is a chromatic subdivision $\text{Div}(\mathcal{I})$ and a color-preserving simplicial map

$$\mu : \text{Div}(\mathcal{I}) \rightarrow \mathcal{O}$$

carried by Γ .

In one direction, the claim is relatively easy. If there exists a read-write protocol, then there exists an immediate snapshot protocol whose protocol complex is $\text{Ch}^N(\mathcal{I})$ for some $N > 0$. The color-preserving simplicial decision map $\delta : \text{Ch}^N(\mathcal{I}) \rightarrow \mathcal{O}$ is the desired map.

The other direction is more difficult: we must show that given a chromatic map $\mu : \text{Div } \mathcal{I} \rightarrow \mathcal{O}$ carried by Div , for some $N > 0$. A protocol can be constructed by solving the convergence task on $\text{Ch}^N(\mathcal{I})$, and then using ϕ as the decision map.

The most straightforward strategy is to show there exists a color-preserving simplicial map

$$\phi : \text{Ch}^N \mathcal{I} \rightarrow \text{Div } \mathcal{I},$$

for some $N > 0$, such that for all $\sigma \in \mathcal{I}$, $\phi(\text{Ch}^N \sigma) \subseteq \text{Div } \sigma$. These maps compose as follows:

$$\text{Ch}^N \mathcal{I} \xrightarrow{\phi} \text{Div } \mathcal{I} \xrightarrow{\mu} \mathcal{O}.$$

These maps can be used to construct a protocol. From an input simplex σ , each process performs the following three steps:

- step 1. execute an N -layer immediate snapshot protocol, halt on a vertex x of the simplicial complex $\text{Ch}^N \sigma$,
- step 2. compute $y = \phi(x)$, yielding a vertex in $\text{Div } \sigma$,
- step 3. compute $z = \mu(y)$, yielding an output vertex.

It is easy to check that all processes halt on the vertexes of a single simplex in $\Gamma(\sigma)$. Moreover, because all maps are color-preserving, each process halts on an output vertex of matching color.

In prior work, the map $\phi : \text{Ch}^N \mathcal{I} \rightarrow \text{Div } \mathcal{I}$ was constructed in the following combinatorial way. The continuous identity map $|\text{Ch}^N \mathcal{I}| \rightarrow |\text{Div } \mathcal{I}|$ has a simplicial approximation $\psi : \text{Ch}^N \mathcal{I} \rightarrow \text{Div } \mathcal{I}$, carried by Δ . The bulk of the proof is concerned with “perturbing” ψ to make it color-preserving, a somewhat long and delicate construction.

Borowsky and Gafni suggested an alternative approach: treat this problem as a task, $(\mathcal{I}, \text{Div } \mathcal{I}, \text{Div})$, in which processes start on vertexes of matching color on a simplex σ of \mathcal{I} , and halt on vertexes of matching color on a single simplex of $\text{Div}(\sigma)$. An explicit protocol that solves this *chromatic simplex agreement* (CSA) task induces the desired map. As noted, the original paper lacked a complete description of the algorithm and a proof, both of which are presented here.

4.1 Chromatic Simplex Agreement

Borowsky and Gafni first consider a more simple task, called non-chromatic simplex agreement (NCSA), in which processes start on the same input complex \mathcal{I} and must converge to any simplex of $\text{Div}(\mathcal{I})$. Processes do not have to land on vertexes of any specified color, though they must remain where they are if they run solo.

NCSA can be solved provided that \mathcal{I} is sufficiently connected. Borowsky and Gafni present a sketch of an inductively constructed protocol, which can be made rigorous via combinatorial topology. While we do not give an explicit protocol here, in the next section we give an explicit protocol for a similar task called *link-based NCSA*, or LNCSA, which is directly used in the construction of the convergence algorithm.

Algorithm 1 shows the Chromatic Simplex Agreement (CSA) protocol, which is round-based. Recall that participating processes begin on a simplex of \mathcal{I} . In the first round, the processes to run a simplex agreement protocol on $\text{Bary}(\text{Div } \mathcal{I})$, so that they collectively choose a nested sequence of simplexes in $\text{Div } \mathcal{I}$. Each process writes its simplex to shared memory and takes an immediate snapshot of the simplexes written by the others. If a process sees a vertex of its color in all simplexes in its snapshot, then the process returns that vertex and finishes. Otherwise, it computes its *view*¹ as the union of all simplexes it saw, minus the vertex of its own color, if it exists. The process proceeds to the next round.

¹ Borowsky and Gafni called the view the “core”.


```

shared participating[ $n+1$ ];
shared views[ $n+1$ ];
shared simplexes[ $n+1, n+1$ ];
protocol chromaticSimplexAgree( $p, v, \mathcal{I}, \text{Div}$ ):
    participating[ $p$ ] :=  $p$ ;
     $s$  := simplexAgree( $v, \mathcal{I}, \text{Div}$ );
    immediate
        simplexes[ $1, p$ ] := ( $s, \emptyset$ );
         $\text{snap}$  := snapshot(simplexes[ $1$ ]);
        if  $\exists u : u \in \bigcap_{t \in \text{snap}} t$  and  $\chi(u) = p$  then
            return  $u$ ;
        else
             $\text{toss} := \{u : u \in \bigcup_{t \in \text{snap}} t \text{ and } \chi(u) = p\}$ ;
             $w := \bigcup_{t \in \text{snap}} t - \text{toss}$ ;
     $r := 2$ ;
    while True do
        immediate
            views[ $r, p$ ] :=  $w$ ;
             $\text{snap}$  := snapshot(views[ $r$ ]);
             $P := \text{snapshot}(\text{participating})$ ;
         $c := \bigcap_{x \in \text{snap}} x$ ;
         $s, \bar{c} := \text{linkNonchromaticSimplexAgree}(v, c, P, \mathcal{I}, \text{Div})$ ;
        immediate
            simplexes[ $r, p$ ] := ( $s, \bar{c}$ );
             $\text{snap}$  := snapshot(simplexes[ $r$ ]);
        if  $\exists u : u \in \bigcap_{t \in \text{snap}} t$  and  $\chi(u) = p$  then
            return  $u$ ;
        else
             $\text{toss} := \{u : u \in \bigcup_{t \in \text{snap}} t \text{ and } \chi(u) = p\}$ ;
             $w := \bigcup_{t \in \text{snap}[0]} t \cup \bigcap_{d \in \text{snap}[1]} d - \text{toss}$ ;
             $r := r + 1$ ;
    end

```

Algorithm 1: The convergence algorithm

A process entering round $r > 1$ has already computed a view. Each process begins round r by writing its view to shared memory and taking immediate snapshots of the shared memory from this round as well as the first round. By retaking a snapshot of the first round, a process updates the *participating set* of processes it observes running the protocol. By taking a snapshot of the views from the current round, the process computes its *core*², defined to be the intersection of the views from the current round's snapshot. Using these two snapshots, the process computes its *convergence complex*, a simplicial neighborhood of the core (defined in the next section). Note that processes typically construct different convergence complexes, though all such complexes will be ordered by inclusion. Each process then chooses a *starting vertex* of its own color in its convergence complex, and runs the LNCSA protocol on the barycentric subdivision of its convergence complex. Processes collectively obtain a nested sequence simplexes, similar to the first round. Each process then replaces its core with the smallest core observed when running LNCSA. Each process writes its simplex and its new core to shared memory and takes a snapshot. If a process sees a vertex of its color in each simplex it saw, it decides on that vertex and finishes. Otherwise, it updates its view and proceeds to the next round.

See Algorithm 1 for pseudocode. In the next section, we give more detailed explanations about how each piece of data is computed.

² Borowsky and Gafni called this the “intersection of cores”.

5 Correctness of the Convergence Algorithm

5.1 Preliminaries

If each process executes Protocol 1, it will decide on a vertex of its color in a simplex on $\text{Div}(\mathcal{I})$, thus solving CSA over \mathcal{I} . In this section we show that the protocol terminates, and that processes choose valid outputs.

Here is some notation useful to prove correctness of the convergence algorithm. Let p_1, \dots, p_{n+1} be the processes participating in the convergence algorithm. Let p be any such process. For each round $r > 1$, p 's state consists of the following: its *participating set* P_p^r , its *core* c_p^r , its *convergence complex* \mathcal{C}_p^r , its *starting vertex* v_p^r , its *simplex* s_p^r , and its view w_p^r .

Here is how each component is computed during round r of the convergence algorithm:

1. The *participating set* P_p^r is the set of processes and corresponding input vertexes that process p sees when it takes a snapshot of the first round's shared memory.
2. The *core* c_p^r is the set of vertexes that may be decision values of processes other than p that have finished the protocol. It is defined to be

$$c_p^r = \bigcap_{j \in J} w_{p_j}^r,$$

where J is the index set of processes seen by p in its snapshot of views from the current round r . When running LNCSA, the process may see smaller cores, in which case p recomputes its core as

$$\bar{c}_p^r = \bigcap_{k \in K} c_{p_k}^r$$

where I is the index set of processes seen by p during LNCSA.

3. The *convergence complex* \mathcal{C}_p^r is the complex on which p runs the LNCSA protocol to choose a new simplex consistent with current decision values. It is computed as

$$\mathcal{C}_p^r = \text{Lk}\left(\bigcap_{k \in K} c_{p_k}^r, \bigcup_{k \in K} P_{p_k}^r\right).$$

4. The *starting vertex* v_p^r is the vertex on which p begins running the LNCSA protocol. Any vertex from \mathcal{C}_p^r with the same color as p may be chosen.
5. The *simplex* s_p^r is the simplex in \mathcal{C}_p^r which p chooses as a result of running the LNCSA protocol. They collectively represent the domain of values from which processes may choose decision values in round r .
6. The *view* w_p^r is the set of vertexes that p sees in round r . It is computed as

$$w_p^r = \bigcup_{i \in I} s_{p_i}^r \cup \bigcap_{i \in I} \bar{c}_{p_i}^r - \{u_p^r\}$$

where I is the index set of processes seen by p in its snapshot of simplexes and cores, and u_p^r is the vertex with the same color as p , if it exists.

5.2 Link-based non-chromatic simplex agreement

Before we can define LNCSA, we need a few technical lemmas.

Lemma 5.1. All views and cores are simplexes.

Proof. By induction. It is clear that views from round 1 are simplexes, since they are all subsets of the largest simplex chosen during simplex agreement. The cores computed in round 2 are also simplexes, since they are intersections of the views from round 1.

Inductively assume that all views from round r are simplexes. Clearly all cores c_p^{r+1} in round $r+1$ are simplexes as well, since they are intersections of views from round r . Views in round $r+1$ are computed as

$$w_p^{r+1} = \bigcup_{i \in I} s_{p_i}^{r+1} \cup \bigcap_{i \in I} \bar{c}_{p_i}^{r+1} - \{u_p^{r+1}\}$$

Fix process p . We know that

$$\bigcup_{i \in I} s_{p_i}^{r+1} = s_{p_\ell}^{r+1}$$

for some $\ell \in I$, since the simplexes are ordered by inclusion. Then $s_{p_\ell}^{r+1}$ is the largest simplex seen by p in its snapshot. Furthermore, we know that $s_{p_\ell}^{r+1} \cup \bar{c}_{p_\ell}^{r+1}$ is a simplex, by definition of the link of a simplex. But

$$\bigcap_{i \in I} \bar{c}_{p_i}^{r+1} \subseteq \bar{c}_{p_\ell}^{r+1}$$

so

$$w_p^{r+1} = \bigcup_{i \in I} s_{p_i}^{r+1} \cup \bigcap_{i \in I} \bar{c}_{p_i}^{r+1} - \{u_p^{r+1}\} \subseteq s_{p_\ell}^{r+1} \cup \bar{c}_{p_\ell}^{r+1}.$$

By downward closure, we conclude that w_p^{r+1} is a simplex.

By induction, all views and cores are simplexes. \square

The next lemma states that the convergence complexes of the processes are ordered by inclusion, ensuring that the NCSA tasks solved by different processes are coherent, even though they may have different convergence complexes.

Lemma 5.2. The convergence complexes of participating processes for any given round are ordered by inclusion.

Proof. Fix round r . If exactly one or fewer processes have not decided, then the claim is trivial. So let p_1 and p_2 be distinct processes that have not decided. Let \mathcal{C}_i denote the convergence complex for p_i . We must show \mathcal{C}_1 and \mathcal{C}_2 are comparable. Without loss of generality, suppose p_1 takes a snapshot of the view arrays for rounds 1 and k before p_2 . Then p_2 sees more views than p_1 , so it computes a larger intersection for its core, meaning that $c_2 \subseteq c_1$, where c_i is the core of p_i in the current round. Furthermore, if Q_i is the set of processes that p_i sees in round 1, then $Q_1 \subseteq Q_2$. Since the link operator is order reversing in the first argument, and order preserving in the second, we have $\text{Lk}(c_1, Q_1) \subseteq \text{Lk}(c_2, Q_2)$. We conclude that $\mathcal{C}_1 \subseteq \mathcal{C}_2$, which proves the lemma. \square

Since convergence complexes are ordered by inclusion, cores and participating sets are ordered in the same way.

The next lemma allows each process to pick a vertex of its own color in its convergence complex when it starts LNCSA.

Lemma 5.3. If process p has not decided by round r , then there is at least one vertex of its color in \mathcal{C}_p^r .

Proof. It suffices to show that no vertex in \mathcal{C}_p^r has the same color as p . The core c_p^r is computed as the intersection of views that p sees in its snapshot of the view array in round r . This includes w_p^{r-1} . But w_p^{r-1} cannot contain a vertex of color p , since any such vertex is explicitly removed in the computation of w_p^{r-1} . Since w_p^{r-1} does not contain a vertex of color p , neither does c_p^r . since \mathcal{I} is chromatic, \mathcal{C}_p^r must contain at least one vertex of color p . \square

The next lemma states that a decision value reached in a given round is contained in all cores and views of subsequent rounds.

Lemma 5.4. If a process decides on a vertex, then that vertex is contained in the views and cores of all processes in all subsequent rounds.

Proof. Suppose process p decides on vertex u in round r_p . We proceed by induction. As the base case, we show that every view computed in round r_p contains u . Let p' be any other process that does not decide this round. There are two cases: when p' computes its new view, either p' sees the simplex that p wrote, or p' does not. In the first case, p' sees what p wrote, which is a simplex σ that must contain u , otherwise p could not decide this round, since p must have seen u in its own simplex. So p' includes σ in the computation of its view, so the view of p' contains u . Now consider the second case, where p' does not see the simplex that p wrote. But for p to decide on u , u must have been contained in all previously written simplexes, including the simplex of p' . Since this simplex contains u , the view of p' will also contain u . In either case, the view of p' contains u .

Now, inductively assume that all views from round $r \geq r_p$ contain vertex u . We want to show that all views from round $r + 1$ also contain u . First, consider the cores computed in round $r + 1$. As intersections of views from round r , all cores computed in round $r + 1$ also contain u . Furthermore, each process p' computes its view in round $r + 1$ as

$$w_{p'}^{r+1} = \bigcup_{i \in I} s_{p_i}^{r+1} \cup \bigcap_{i \in I} \bar{c}_{p_i}^{r+1} - \{u_{p'}^r\}.$$

Since each core contains u , we have

$$u \in \bigcap_{i \in I} \bar{c}_{p_i}^{r+1} \subseteq w_{p'}^{r+1}$$

so the view of p' computed in round $r + 1$ also contains u . Therefore all views in round $r + 1$ also contain u . By induction, all views in all rounds after r_p contain u . Since cores are computed as intersections of views, all cores in rounds $r > r_p$ also contain u . \square

We formally define LNCSA and give a protocol similar to that of NCSA.

Lemma 5.5. Fix round r , and suppose processes have computed their cores. Then there is a wait-free immediate-snapshot protocol for converging to a simplex on $\text{Lk}(\bigcap_{k \in K} c_{pk}^r, \bigcup_{k \in K} P_{pk}^r)$.

Proof. Let $(\mathcal{I}, \text{Div}(\mathcal{I}), \text{Div})$ be the chromatic simplex agreement task we want to solve via the convergence algorithm. Fix round r , and suppose the processes have just computed their links. They will next collectively converge to a simplex that is consistent with the smallest core among them. That is, they will converge on the largest link among the processes. This is the LNCSA task.

We formally define the task. Each process's state is defined by its core c , its participating set p , and the starting vertex v it chooses in its link (which by Lemma 5.3 it can do), so the input complex \mathcal{A} of LNCSA has vertex set consisting of triples (v, c, P) such that $v \in P$ and $\text{Car}(c, \mathcal{I}) \subseteq P$. A set of triples $\{(v_i, c_i, P_i)\}$ is a simplex in \mathcal{A} if the v_i all have distinct colors and $\{c_i\}$ and $\{P_i\}$ are ordered by inclusion. We require the v_i to have different colors since each process chooses a vertex of its own color, and we require $\{c_i\}$ and $\{P_i\}$ to be ordered in the same way by inclusion, since they were computed using snapshots.

So \mathcal{A} is a subcomplex of

$$\Delta(V) \times \text{Bary}(\text{Div}(\mathcal{I})) \times \text{Bary}(\mathcal{I}),$$

where V is the vertex set of \mathcal{I} . The output complex of LNCSA is $\mathcal{B} = \text{Bary}(\text{Div}(\mathcal{I}))$, since each process chooses a simplex on $\text{Div}(\mathcal{I})$. Processes that execute in isolation stay where they are. Otherwise, processes converge to a simplex in the largest link they see, so if $\sigma = \{(v_i, c_i, P_i)\}$ is a simplex in \mathcal{A} , then we have $\Gamma(\sigma) = \{v\}$ if $\sigma = \{(v, c, P)\}$, and $\Gamma(\sigma) = \text{Lk}(\bigcap c_i, \bigcup P_i)$. By Lemma 5.2, Γ is monotonic, so it is a carrier map.

Thus LNCSA is the task $(\mathcal{A}, \mathcal{B}, \Gamma)$. We show that this task is solvable by constructing a continuous

$$f : |\mathcal{A}| \rightarrow |\mathcal{B}|$$

carried by Γ . We induct on skeletons to construct an such an f by defining carrier-preserving

$$f^m : |\text{skel}^m(\mathcal{A})| \rightarrow |\mathcal{B}|$$

for each m . As base case, we define f^0 as $f^0((v, c, P)) = v$ for each vertex (v, c, P) of \mathcal{A} . The function f^0 is clearly continuous and carried by Γ . Now inductively assume we have defined f^m . We want to define

$$f^{m+1} : |\text{skel}^{m+1}(\mathcal{A})| \rightarrow |\mathcal{B}|.$$

Let $\{\sigma_i\}$ be the facets of $\text{skel}^{m+1}(\mathcal{A})$, and for each i , let $\sigma_i = \{(v_{ij}, c_{ij}, P_{ij})\}_{j \leq m+2}$. Let

$$\bar{c}_i = \bigcap_j c_{ij} \quad \text{and} \quad \bar{P}_i = \bigcup_j P_{ij}.$$

By the inductive hypothesis, f^m is carried by Γ , which implies that the image of f^m is contained in

$$\mathcal{L}_i = \text{Lk}(\bar{c}_i, \bar{P}_i).$$

We also know that $\dim(\mathcal{L}_i) \geq \dim(\sigma_i) = m$, since each $v_{ij} \in \mathcal{L}_i$, each v_{ij} has different color, and \mathcal{L}_i is chromatic. Suppose $\dim(\bar{P}_i) = n$. We know that \mathcal{L}_i is a link in the subdivided simplex $\text{Bary}(\text{Div}(\bar{P}_i))$, which is link-connected, so since $\dim(\mathcal{L}_i) \geq m$, we conclude that \mathcal{L}_i is at least $n-2-(n-m-1) = (m-1)$ -connected. Using this, along with the fact that $\text{Im}(f^m) \subseteq \mathcal{L}_i$, we can extend $f^m|_{\partial\sigma_i}$ to a function $f_i^{m+1} : |\sigma_i| \rightarrow |\mathcal{L}_i|$. Next, using the pasting lemma, we can glue the f_i^{m+1} together to obtain a map

$$f^{m+1} : |\text{skel}^{m+1}(\mathcal{A})| \rightarrow \mathcal{B}$$

extending f^m . By construction, f^{m+1} is carried by Γ . By induction, we have obtained a continuous function $f : |\mathcal{A}| \rightarrow |\mathcal{B}|$. By the simplicial approximation theorem, the task $(\mathcal{A}, \mathcal{B}, \Gamma)$ is wait-free solvable using immediate snapshots. \square

5.3 Termination and Validity

Next, we show that all processes eventually decide.

Theorem 5.6. All processes participating in the convergence algorithm eventually decide.

Proof. We show that at least one process decides each round. Fix a round, and let $\{p_0, \dots, p_\ell\}$ be the set of participating processes. The processes each run LNCSA protocols over barycentric subdivisions of the convergence complexes, and by Lemma 5.5, they converge to a simplex τ on the largest subcomplex, say $\text{Bary}(\mathcal{C})$. By definition of Bary, the simplex $\tau \in \text{Bary}(\mathcal{A}_{i_k})$ corresponds to a chain of simplexes $\sigma_{i_0} \subseteq \dots \subseteq \sigma_{i_\ell}$. Their intersection is σ_{i_0} , which is clearly nonempty, so choose a vertex $\hat{v} \in \sigma_{i_0}$. The color of \hat{v} , denoted as $\chi(\hat{v})$, cannot be the color of any nonparticipating process, because all subcomplexes $\text{Bary}(\mathcal{C})$ are all subcomplexes of the processes' carrier, which contains only vertexes whose colors are those of the participating processes, since \mathcal{I} is chromatic. Neither can $\chi(\hat{v})$ be the color of a process that decided in a previous round. Supposing not, this means the largest convergence complex contains vertexes of color $\chi(\hat{v})$, meaning that the corresponding core could not contain a vertex of color $\chi(\hat{v})$, since the link contains vertexes of color exactly those not in the corresponding core. This contradicts Lemma 5.4, since decided vertexes must be contained in all cores.

Let $\hat{p} \in P$ be the process whose color is $\chi(\hat{v})$, and let $K \subseteq \{0, \dots, \ell\}$ be the index set of processes that \hat{p} saw during its snapshot of the simplex array. Then

$$\bigcap_{k \leq \ell} \sigma_{i_k} \subseteq \bigcap_{k \in K} \sigma_{i_k},$$

so $\hat{v} \in \bigcap_{k \in K} \sigma_{i_k}$. Since \hat{p} sees \hat{v} in all simplexes in its snapshot, \hat{p} decides on \hat{v} . It follows that at least one process decides in each round, so all processes decide in at most $n+1$ rounds. \square

Theorem 5.7. Participating processes converge to a simplex in their carrier.

Proof. We show that for each round r , the set of vertexes that processes have decided form a simplex. We argue by induction, starting at round 1. In round 1, it is clear that decision values form a simplex, since processes decide on vertexes from the largest simplex chosen by running LNCSA. All decision values in round 1 must be contained in this largest simplex, hence they must form a simplex.

Fix round r , and inductively assume that the vertexes which processes decided during the rounds up to r form a simplex, which we call τ_r . Consider round $r+1$. Processes in this round choose vertexes on a simplex of the barycentric subdivision of the largest convergence complex, $\text{Bary}(\mathcal{C})$, which is determined by the smallest core. So vertexes on which processes decide this round are contained in an ascending chain

of simplexes in \mathcal{C} . Call the largest such simplex Σ_{r+1} . Let c_{r+1} be the smallest core, corresponding to the largest convergence complex. Then by definition of the link, $c_{r+1} \cup \Sigma_{r+1}$ is a simplex in \mathcal{C} , since the largest link is determined by simplex c_r . From Lemma 5.4, we must have $\tau_r \subseteq c_{r+1}$, so $\tau_r \cup \Sigma_{r+1}$ is also a simplex in \mathcal{C} , by downward closure of simplicial complexes. Let $\sigma_{r+1} \subseteq \Sigma_{r+1}$ be the set of all vertexes that processes on decide during round $r + 1$. Again by downward closure,

$$\tau_r \cup \sigma_{r+1} \subseteq \tau_r \cup \Sigma_{r+1},$$

so $\tau_r \cup \sigma_{r+1}$ is a simplex. But by definition of τ_r , $\tau_{r+1} = \tau_r \cup \sigma_{r+1}$, where τ_{r+1} is exactly the set of vertexes on which processes have decided up to round $r + 1$. So τ_{r+1} is also a simplex. By induction, it follows that the processes' decision values form a simplex.

Processes can choose decision only values in their carrier, since all links are computed relative to the observed participating set, and all decision values are chosen from these links. This completes the proof of the theorem, and the proof of correctness of the convergence algorithm. \square

6 Application to more general tasks

Recall that CSA over a chromatic subdivision $\text{Div}(\mathcal{K})$ is defined as the task $(\mathcal{K}, \text{Div}(\mathcal{K}), \text{Div})$. The proof of the convergence algorithm shows that this task has a wait-free read-write protocol. To make the convergence algorithm work, we required that $\text{Div}(\mathcal{K})$ be link-connected so that processes can iteratively converge over ever smaller subcomplexes. Phrased in a different way, the convergence algorithm allows us to find a chromatic simplicial map

$$\phi : \text{Ch}^N(\mathcal{I}) \rightarrow \text{Div } \mathcal{I}$$

carried by Div . Given the continuous (identity) map $\text{id} : |\mathcal{I}| \rightarrow |\text{Div}(\mathcal{I})|$ also carried by Div . In fact, link-connectivity is the essential property of the output complex. In particular, the the convergence algorithm may be applied to more general continuous functions $f : |\mathcal{I}| \rightarrow |\mathcal{O}|$ carried by some Γ . Given the assumption that $\Gamma(\sigma)$ is link-connected for all $\sigma \in \mathcal{I}$, we can use the convergence algorithm to obtain a chromatic simplicial map $\phi : \text{Ch}^N(\mathcal{I}) \rightarrow \mathcal{O}$ also carried by Γ . We state this observation as a theorem.

Theorem 6.1. Let $f : |\mathcal{I}| \rightarrow |\mathcal{O}|$ be a continuous map between chromatic complexes and let $\Gamma : \mathcal{I} \rightarrow \mathcal{O}$ be a carrier map such that $\Gamma(\sigma)$ is link-connected for each $\sigma \in \mathcal{I}$. Suppose f is carried by Γ . Then there exists a chromatic, carrier-preserving simplicial map $\phi : \text{Ch}^N(\mathcal{I}) \rightarrow \mathcal{O}$.

7 Related work

A more complete treatment of topological models for concurrent computing can be found in the textbook of Herlihy, Kozlov, and Rajsbaum [11].

The original ACT [13, 12] used combinatorial arguments to construct the color-preserving map required by the theorem. Borowsky and Gafni [5] proposed the alternative algorithmic approach to constructing this map, but without complete definitions or a proof of correctness. Guerraoui and Kuznetsov [10] compare the two approaches. Gafni *et al.* [9] recently generalized the ACT to encompass infinite executions and other models.

The first application of the ACT was to prove the impossibility of the k -set agreement task [7], a result also proved, using other techniques, by Borowsky and Gafni [4], and by Saks and Zaharoglou [16]. These results generalize the classic proofs of the impossibility of consensus due to Fischer *et al.* [8] and Biran *et al.* [3].

Castaneda and Rajsbaum [6] use the ACT to show that the *renaming* task [1] for $n + 1$ processes has no wait-free read/write protocol with $2n$ output names when $n + 1$ is a prime power, but that a protocol does exist when $n + 1$ is not a prime power. Attiya *et al.* [2] and Kozlov [14] give upper bounds on the running times of such protocols.

8 Remarks

The heart of the ACT is the construction of a chromatic map from a chromatic subdivision of the input complex to the output complex. While it is easy to construct a non-chromatic map using the well-known *simplicial approximation theorem* [15, p.89], the only prior construction [11, 12] was long and complex. The algorithmic approach proposed by Borowsky and Gafni [4] was intuitively appealing, but lacked a complete statement of the algorithm and a proof. Here, we have given such a statement and proof, and we believe the result, while far from simple, is easier to follow, and yields new insight into this key construction.

References

- [1] Hagit Attiya, Amotz Bar-Noy, Danny Dolev, Daphne Koller, David Peleg, and Rudiger Reischuk. Achievable cases in an asynchronous environment. pages 337–346, October 1987.
- [2] Hagit Attiya, Armando Castañeda, Maurice Herlihy, and Ami Paz. Upper bound on the complexity of solving hard renaming. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 190–199, New York, NY, USA, 2013. ACM.
- [3] O. Biran, S. Moran, and S. Zaks. A combinatorial characterization of the distributed 1-solvable tasks. *Journal of Algorithms*, 11:420–440, 1990.
- [4] Elizabeth Borowsky and Eli Gafni. Generalized FLP impossibility result for t -resilient asynchronous computations. May 1993.
- [5] Elizabeth Borowsky and Eli Gafni. A simple algorithmically reasoned characterization of wait-free computations. In *Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing*, pages 189–198, August 1997.
- [6] Armando Castañeda and Sergio Rajsbaum. New combinatorial topology upper and lower bounds for renaming. In *PODC '08: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing*, pages 295–304, New York, NY, USA, 2008. ACM.
- [7] Soma Chaudhuri. Agreement is harder than consensus: set consensus problems in totally asynchronous systems. pages 311–234, August 1990.
- [8] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, June 1982.
- [9] Eli Gafni, Petr Kuznetsov, and Ciprian Manolescu. A generalized asynchronous computability theorem. In *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, pages 222–231, 2014.
- [10] Rachid Guerraoui and Petr Kuznetsov. Two faces of the asynchronous computability theorem. In *The 4th Workshop in Geometry and Topology in Concurrency and Distributed Computing*, 2004.
- [11] Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology*. Elsevier, 2013.
- [12] Maurice Herlihy and Nir Shavit. The topological structure of asynchronous computability. *J. ACM*, 46(6):858–923, 1999.
- [13] Maurice P. Herlihy and Nir Shavit. The asynchronous computability theorem for t -resilient tasks. pages 111–120, May 1993.
- [14] Dmitry N. Kozlov. Structure theory of flip graphs with applications to weak symmetry breaking. *CoRR*, abs/1511.00457, 2015.
- [15] J. R. Munkres. *Elements Of Algebraic Topology*. Addison Wesley, Reading MA, 1984.
- [16] Michael Saks and Fotis Zaharoglou. Wait-free k -set agreement is impossible: The topology of public knowledge. May 1993.